Apple Assembly Line

$1.80

Volume 7 -- Issue 7 April, 1987

In This Issue...

Bytestalker Software Analyzer Card

Old-fashioned computers always came with a front panel, loaded with lights and switches. You could usually stop a program, single-step it, and even set a breakpoint address to halt execution at a particular address. Personal computers don't include front panels, but now you can add part of one to your Apple. J. Stuart Enterprises, Box 310, Grass Valley, CA 95945, is offering the Bytestalker Software Analyzer Card. The card includes 16 switches for selecting a breakpoint address, a three-position mode switch (run, stop, trap), and a two position step-hold switch. There are also six single hex digit displays, to show you address and contents. You can choose single-stepping at the machine cycle level, or the instruction level. The price is only $88.50, plus postage and insurance. Contact J. Stuart at the above address or at phone (916)273-9188.

We're Gaining on it...

We've gained about a week in our efforts to get AAL back on schedule: it looks like the April issue will actually get to the printer during April! Next month we'll be shooting to get the May issue in the mail before the end of May, and soon you'll be receiving each month's issue during the appropriate month. Thanks for your patience.

80-Column Screen Display Subroutine......Bob Sander-Cederlof

In the very first issue of AAL, back in October 1980, I wrote an article about an interpretive screen display subroutine. A lot of things have happened since 1980! Although perhaps it is insignificant compared to the rise and fall of nations, one change which dramatically affects Apple programmers is that now almost everything is done on an 80-column screen. That old message printer was tied to the 40-column screen, so I think an upgrade is in order.

The following program has more features than the 1980 version, and is also easier to use. Another nice change since 1980 is that most of the assemblers available for the Apple now include a macro capability. Macros make it easier to use special coded sequences of bytes like those necessary in the message printer.

Where is the message?

There are many different ways to tell a message-printing subroutine where the message to be printed is located in memory. The method I used in my 1980 article was to assign a message number to each message, and pass that number in the A-register; the message printer used the message number as an index to a table of message addresses. Sometimes I have used the message number technique without a table of message addresses: I put all the messages together in one consecutive area of memory; the message printer searched through the messages counting end-of-message characters until it found the N-th message, and then started printing.

I show two other techniques in the program which follows. The first one involves loading the full address of the message to be printed into the A- and Y-registers. In my program I put the high-byte of the address in the Y-register, and the low-byte in the A-register. Of course you could use any registers any way you wish, but this is what I used. The code for MESSAGE.PRINTER is shown in lines 1250-1380. I defined a macro named "MSG" in lines 1190-1230 to make it easier to write the code to call the message printer. Calls to the message printer may be written simply as one-line macro calls, such as:

```
               >MSG TITLE
               >MSG COPYRIGHT
```

where TITLE and COPYRIGHT are the names of specific messages assembled elsewhere in a program.

The other technique I show here is probably the most popular one, if the programs I have disassembled are any indication. In this method the text of the message to be printed immediately follows the JSR to the message printer. The message printer uses the return address placed on the stack by the JSR to locate the message. After the message is printed, the return address is modified so that it points to the next instruction after the message before RTSing. ANOTHER.MESSAGE.PRINTER is shown in lines 1420-1640.

S-C Macro Assembler Version 2.0.....DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners...................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Source Code of S-C Macro 2.0 (DOS or ProDOS).......each, additional $100
S-C Disassembler (ProDOS only)...without source code $30, with source $50
RAK-Ware DISASM (DOS only)......without source code $30, with source $50
ProVIEW (ProDOS only)..$20
Full Screen Editor for S-C Macro (with complete source code).........$49
S-C Cross Reference Utility.....without source code $20, with source $50
S-C Word Processor (with complete source code).......................$50
DP18 and DPFP, including complete source and object code.............$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
Bag of Tricks 2 (Quality Software)........................($49.95) $45 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $50 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)
Beagle Bros. Applesoft Compiler (ProDOS only)...............($74.95) $65 *

AAL Quarterly Disks.......................each $15, or any four for $45
Each disk contains the source code from three issues of AAL:
Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler. Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 25 for $15 *
Vinyl disk pages, 6"x8.5", hold two disks each.................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100 *
Envelopes for Diskette Mailers........................... 6 cents each

65802 Microprocessor (Western Design Center)..................($95) $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG)...($149.50) $140 *
Minuteman 250 Uninterruptible Power Supply..................($359) $320 +

"Apple //gs Technical Reference", Fischer..................($19.95) $19 *
"65816/65802 Assembly Language Programming", Fischer........($21.95) $20 *
"Programming the 65816", Eyes...............................($22.95) $21 *
"Apple //e Reference Manual", Apple Computer................($24.95) $23 *
"Apple //c Reference Manual", Apple Computer................($24.95) $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95) $27 *
"Apple ProDOS: Advanced Features for Programmers", Little..($17.95) $17 *
"ProDOS Inside and Out", Doms & Weishaar...................($16.95) $16 *
"DOStalk Scrapbook", Weishaar & Kersey.....................($14.95) $14 *
"Beneath Apple ProDOS", Worth & Lechner....................($19.95) $18 *
"Beneath Apple DOS", Worth & Lechner.......................($19.95) $18 *
"Inside the Apple //c", Little.............................($19.95) $18 *
"Inside the Apple //e", Little.............................($19.95) $18 *
"Understanding the Apple //e", Sather......................($24.95) $23 *
"Understanding the Apple II", Sather.......................($22.95) $21 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95) $18 *
"Apple][Circuit Description", Gayler.....................($22.95) $21 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95) $18 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95) $18 *
"Enhancing Your Apple II, vol. 1", Lancaster...............($15.95) $15 *
"Enhancing Your Apple II, vol. 2", Lancaster...............($17.95) $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95) $20 *
"Microcomputer Graphics", Myers............................($14.95) $14 *
"Assembly Lines -- the Book", Wagner.......................($19.95) $18 *
"AppleVisions", Bishop & Grossberger.......................($39.95) $36 *

* On these items add $2.00 for the first item and
$.75 for each additional item for US shipping.
+ Inquire for shipping cost.
Foreign customers inquire for postage needed.
Texas residents please add 7.25 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** Master Card, VISA, Discover and American Express ***

Both message printers begin by storing the address of the message into a page-zero pointer. Then a subroutine named GET.NEXT.CHAR in lines 1680-1750 is called to pick up characters of the message. Since the last instruction in this subroutine is the one which picks up the character, the Zero-status will be set if the $00 byte indicating end-of-message is picked up.

What is in the Message?

The real intelligence of my message printers is in the subroutine PUT.CHAR, starting at line 1760. We have already mentioned that a $00 byte indicates end-of-message. PUT.CHAR expects the X-register to contain a repeat count, and the A-register to contain the character to be printed. If the byte in the A-register has a value between $40 and $FF, it is sent to COUT to be printed (X) times. However, if the byte in the A-register has a value between $01 and $3F, it is interpreted in special way.

Values between $80 and $9F are control characters, and will be interpreted by the monitor firmware. If you are in an 80-column //e, //c, or //gs, the interpretation will be as shown in lines 1850-2070. In 40-column mode not all of the functions are active. A Viewmaster or other type of 80-column card will have its own set of controls.

Values between $01 and $18 are interpreted by PUT.CHAR as the first byte of a two-byte cursor position code. This first byte indicates the line number (1-24 decimal) to put the cursor on, and the following byte is the column number (0-79 decimal). I defined a macro "POS" in lines 2100-2130 to simplify putting position codes into messages.

Values between $19 and $3F are used to index a special function table. My table is shown in lines 2700-2840. Notice that I used a macro "FUN" to write each entry in the table. This macro defines the special function code as "F.name", and assembles the address-1 of the code which performs the function. I thought of seven handy functions, but you can probably think of a lot more. As you do, just add >FUN lines to the table and write subroutines to perform the functions. Use mine as models, and you should be successful.

The methods I used for setting up the cursor position and handling other special functions work with Apple's firmware in the 40- and 80-column modes; if you have a Viewmaster or other 80-column card, you will need to make some changes in PUT.CHAR and its subroutines.

I made macros for the all of the functions, shown in lines 2140-2460. You might also find it handy to define macros for some of the control-characters processed by the 80-column firmware, but I didn't do it here.

Even though I have the "POS" function for positioning the cursor on the screen, I still wanted the ability to do HTAB and VTAB independently. These are my first two functions. By

writing ">HTAB xx" you can position to column xx on the current line. ">VTAB yy" will put the cursor on line yy without changing the column. Again, yy is a number between 1 and 24, and xx runs from 0 to 79 (decimal).

If you remember, normal characters are printed N times according to the value in the X-register when you call PUT.CHAR. The REP function lets you set up the repeat count. ">REP n,char" will cause char to be printed n times.

The CEN funtion is perhaps the most powerful one. It centers a string on the current line. You simply write something like

    >CEN "This string will be centered"

...and it will be. The code for this function is in lines 3020-3180. All characters which follow the CEN function code are counted, up to the next function code or control character. Then the position for the first character is computed by halving the difference between window width and string length, and an HTAB performed.

FIL and FCL will fill an entire line with a specified character. FIL lets you say which line, and FCL fills the current line. You could do the same thing by putting the cursor at the beginning of the line to be filled and then using a repeat count larger than the window width followed by the fill character, but this way is easier. Let the computer work for you!

WIN lets you specify a window on the screen. This is very rudimentary as window routines go, because all it does is load the four locations the monitor uses to define the window: left, width, top, and bottom. "Left" is the column number to start the window at, and is a value from 0 to 79. "Width" is the number of characters in a line, and has a value from 1 to 80. Just be sure that these two numbers do not add to a value greater than 80, unless you really want to clobber your program. "Top" is the line number of the top of the window, having a value from 0 to 23. "Bottom" is the line number of the bottom line of the window, plus 1; the value goes from 1 to 24. The full 80-column window would be set up with ">WIN 0,80,0,24".

How about some examples?

I wrote two sample programs. "T", in lines 3500-3540, calls MESSAGE.PRINTER to print the messages in lines 3560-3790. "TT", in lines 3950-4170, calls ANOTHER.MESSAGE.PRINTER to print embedded messages.

```
                  1010 *SAVE S-C.MSG.MACHINE
                  1020 *--------------------------------
20-               1030 MON.LEFT   .EQ $20
21-               1040 MON.WIDTH  .EQ $21
22-               1050 MON.TOP    .EQ $22
23-               1060 MON.BOTTOM .EQ $23
24-               1070 MON.CH     .EQ $24
25-               1080 MON.CV     .EQ $25
                  1090 *--------------------------------
057B-             1100 MON.CH80   .EQ $57B
                  1110 *--------------------------------
```

```
FC22-                  MON.VTAB   .EQ $FC22
FDED-                  MON.COUT   .EQ $FDED
                       *--------------------------------
00-                    MSG.PNTR   .EQ $00,01
                       *--------------------------------
                       *   Address of message in Y,A registers
                       *--------------------------------
                              .MA MSG     >MSG <message address>
                              LDA #]1
                              LDY /]1
                              JSR MESSAGE.PRINTER
                              .EM
                       *--------------------------------
                       MESSAGE.PRINTER
0800- 85 00                   STA MSG.PNTR       Store message address
0802- 84 01                   STY MSG.PNTR+1         into page-zero pointer
0804- 8A                      TXA                Save X-register
0805- 48                      PHA
0806- A2 01                   LDX #1             Clear repetition counter
0808- 20 3F 08                JSR GET.CHAR       First char of message
080B- F0 08                   BEQ .2             ...empty message!
080D- 20 44 08         .1     JSR PUT.CHAR       Process the character
0810- 20 39 08                JSR GET.NEXT.CHAR     get another one
0813- D0 F8                   BNE .1             ...not at end of message yet
0815- 68               .2     PLA                Restore X-register
0816- AA                      TAX
0817- 60                      RTS                Return to caller
                       *--------------------------------
                       *   Message follows JSR to message printer
                       *--------------------------------
                       ANOTHER.MESSAGE.PRINTER
0818- 68                      PLA                Store message address (-1)
0819- 85 00                   STA MSG.PNTR          into page-zero pointer
081B- 68                      PLA
081C- 85 01                   STA MSG.PNTR+1
081E- 8A                      TXA                Save X-register
081F- 48                      PHA
0820- 98                      TYA                Save Y-register
0821- 48                      PHA
0822- A2 01                   LDX #1             Clear repetition counter
0824- D0 03                   BNE .2             ...Always
0826- 20 44 08         .1     JSR PUT.CHAR       Process a character
0829- 20 39 08         .2     JSR GET.NEXT.CHAR
082C- D0 F8                   BNE .1             ...not at end of message yet
082E- 68                      PLA                Restore Y-register
082F- A8                      TAY
0830- 68                      PLA                Restore X-register
0831- AA                      TAX
0832- A5 01                   LDA MSG.PNTR+1     Get return address back on stack
0834- 48                      PHA
0835- A5 00                   LDA MSG.PNTR
0837- 48                      PHA
0838- 60                      RTS                Return to caller
                       *--------------------------------
                       *   Get a message character
                       *--------------------------------
                       GET.NEXT.CHAR
0839- E6 00                   INC MSG.PNTR
083B- D0 02                   BNE GET.CHAR
083D- E6 01                   INC MSG.PNTR+1
                       GET.CHAR
083F- A0 00                   LDY #0
0841- B1 00                   LDA (MSG.PNTR),Y
0843- 60                      RTS
                       *--------------------------------
                       *   Process a message character
                       *
                       *      $00      END OF MESSAGE
                       *      $YY XX   where is between $01 and $18, position
                       *                  the cursor as vtab YY, htab XX.
                       *      $19-3F    special functions
                       *      $40-FF    print as is
                       *
                       *   Characters between $80 and $9F are interpreted by
                       *   the Apple 80-column firmware as follows:
                       *
                       *       $87       bell
                       *       $88       backspace
                       *       $8A       move cursor down one line
```

```
                    *           $8B      clear from cursor to end of window
                    *           $8C      clear window and home cursor
                    *           $8D      carriage return
                    *           $8E      normal video
                    *           $8F      inverse video
                    *           $91      switch to 40-column
                    *           $92      switch to 80-column
                    *           $95      de-activate firmware
                    *           $96      scroll display down one line
                    *           $97      scroll display up one line
                    *           $98      disable Mouse Text
                    *           $99      home cursor without clearing window
                    *           $9A      clear current line
                    *           $9B      enable Mouse Text
                    *           $9C      move cursor forward one space
                    *           $9D      clear to end of line
                    *           $9F      move cursor up one line
                    *--------------------------------
                    *   Message Content Macros
                    *--------------------------------
                           .MA POS      Position cursor (vtab, htab)
                           .DA #]1,#]2
                           .EM
                    *--------------------------------
                           .MA REP      Repeat character N times
                           .DA #F.REP,#]1,#"]2
                           .EM
                    *--------------------------------
                           .MA CEN      Center a string on current line
                           .DA #F.CEN
                           .AS -"]1"
                           .EM
                    *--------------------------------
                           .MA STR      normal Ascii string
                           .AS -"]1"
                           .EM
                    *--------------------------------
                           .MA FIL      Fill specified line with character
                           .DA #F.FIL,#]1,#"]2
                           .EM
                    *--------------------------------
                           .MA FCL      Fill current line with character
                           .DA #F.FCL,#"]1
                           .EM
                    *--------------------------------
                           .MA WIN      Define a new text window
                           .DA #F.WIN,#]1,#]2,#]3,#]4
                           .EM
                    *--------------------------------
                           .MA HTAB     Htab the cursor
                           .DA #F.HTAB,#]1
                           .EM
                    *--------------------------------
                           .MA VTAB     Vtab the cursor
                           .DA #F.VTAB,#]1
                           .EM
                    *--------------------------------
                    PUT.CHAR
0844- C9 40                CMP #$40     IS IT PRINTABLE?
0846- 90 08                BCC .2       ...NO, SPECIAL FUNCTION
0848- 20 ED FD      .1     JSR MON.COUT
084B- CA                   DEX          Decrement repeat count
084C- D0 FA                BNE .1       ...repeat the character
084E- E8                   INX          Put the repeat count back to 1
084F- 60                   RTS
                    *---Process function char---------
0850- C9 19         .2     CMP #$19     >POS VV,HH or special function?
0852- 90 1F                BCC P.POS    ...>POS call
                    *---Branch to function code-------
0854- 0A                   ASL          Double function # to make index
0855- A8                   TAY
0856- C0 42                CPY #FUN.TBL.SZ
0858- B0 08                BCS .3       ...Ignore illegal function
085A- B9 32 08             LDA FUN.TBL+1,Y
085D- 48                   PHA          Put function code address-1 on stack
085E- B9 31 08             LDA FUN.TBL,Y
0861- 48                   PHA
0862- 60            .3     RTS          Go to it!
                    *--------------------------------
```

```
                  2700          .MA FUN
                  2710 F.]1     .EQ *-FUN.TBL/2
                  2720          .DA P.]1-1
                  2730          .EM
                  2740 *--------------------------------
0831-             2750 FUN.TBL .EQ *-$19-$19
0863-             2760  >FUN VTAB          Vtab YY
0865-             2770  >FUN HTAB          Htab XX
0867-             2780  >FUN REP           Repeat, count, character
0869-             2790  >FUN CEN           Center, string, null
086B-             2800  >FUN NUL           ...ends centered string
086D-             2810  >FUN FIL           Fill, line #, char
086F-             2820  >FUN FCL           Fill current line, char
0871-             2830  >FUN WIN           Window, left, width, top, bottom
42-               2840 FUN.TBL.SZ .EQ *-FUN.TBL
                  2850 *--------------------------------
0873- 20 BC 08    2860 P.POS    JSR MY.VTAB      Do vertical tab
0876- 20 39 08    2870 P.HTAB JSR GET.NEXT.CHAR ...get horoz value
0879- C5 21       2880          CMP MON.WIDTH    Stay inside current window
087B- 90 04       2890          BCC .1           ...it is inside
087D- A5 21       2900          LDA MON.WIDTH    ...use right edge
087F- E9 01       2910          SBC #1
0881- 85 24       2920 .1       STA MON.CH       40-column horiz pos'n
0883- 8D 7B 05    2930          STA MON.CH80     80-column horiz pos'n
0886- 60          2940          RTS
                  2950 *--------------------------------
0887- 20 39 08    2960 P.REP    JSR GET.NEXT.CHAR
088A- AA          2970          TAX            REPETITION COUNT
088B- 60          2980          RTS
                  2990 *--------------------------------
                  3000 *   Center a string on current line
                  3010 *--------------------------------
088C- A0 00       3020 P.CEN    LDY #0         Count # chars in string
088E- C8          3030 .1       INY            from next char to char below $40
088F- B1 00       3040          LDA (MSG.PNTR),Y       or btwn $80 and $9F
0891- 30 04       3050          BMI .2         80-FF
0893- C9 40       3060          CMP #$40
0895- B0 F7       3070          BCS .1         40-7F, keep counting
0897- C9 A0       3080 .2       CMP #$A0
0899- B0 F3       3090          BCS .1         A0-FF, keep counting
089B- 88          3100          DEY            00-3F or 80-9F, end of string
089C- 98          3110          TYA            (Line.width - string.len) / 2
089D- 49 FF       3120          EOR #$FF         gives starting horiz posn
089F- 38          3130          SEC
08A0- 65 21       3140          ADC MON.WIDTH
08A2- 4A          3150          LSR
08A3- 85 24       3160          STA MON.CH
08A5- 8D 7B 05    3170          STA MON.CH80
08A8- 60          3180 P.NUL    RTS
                  3190 *--------------------------------
08A9- 20 39 08    3200 P.FIL    JSR GET.NEXT.CHAR
08AC- 20 BC 08    3210          JSR MY.VTAB
                  3220 P.FCL
08AF- A9 00       3230          LDA #0
08B1- 85 24       3240          STA MON.CH
08B3- 8D 7B 05    3250          STA MON.CH80
08B6- A6 21       3260          LDX MON.WIDTH
08B8- 60          3270          RTS
                  3280 *--------------------------------
08B9- 20 39 08    3290 P.VTAB JSR GET.NEXT.CHAR
                  3300 MY.VTAB
08BC- C5 23       3310          CMP MON.BOTTOM   Stay inside current window
08BE- 90 02       3320          BCC .1           ...it is inside
08C0- A5 23       3330          LDA MON.BOTTOM   Go to bottom line
08C2- 85 25       3340 .1       STA MON.CV       We use line # 1 to N, but
08C4- C6 25       3350          DEC MON.CV          monitor uses 0 to N-1
08C6- 4C 22 FC    3360          JMP MON.VTAB
                  3370 *--------------------------------
                  3380 P.WIN
08C9- 20 39 08    3390          JSR GET.NEXT.CHAR
08CC- 85 20       3400          STA MON.LEFT
08CE- 20 39 08    3410          JSR GET.NEXT.CHAR
08D1- 85 21       3420          STA MON.WIDTH
08D3- 20 39 08    3430          JSR GET.NEXT.CHAR
08D6- 85 22       3440          STA MON.TOP
08D8- 20 39 08    3450          JSR GET.NEXT.CHAR
08DB- 85 23       3460          STA MON.BOTTOM
08DD- A9 99       3470          LDA #$99
08DF- 4C ED FD    3480          JMP MON.COUT
```

```
                *-------------------------------
08E2-           T       >MSG MSG0
08E9-                   >MSG MSG2
08F0-                   >MSG MSG1
08F7-                   >MSG MSG3
08FE- 60                RTS
                *---------------------------------
08FF- 8C        MSG0    .HS 8C        HOME SCREEN
0900-                   >WIN 10,38,0,24
0905-                   >FIL 1,-            FILL TOP LINE WITH DASHES
0908- FC                .HS FC              ONE BAR ON NEXT LINE
0909- 8D 88 FC
090C- FC                .HS 8D.88.FC.FC     2 BARS
090D- 8D 88 FC
0910- FC                .HS 8D.88.FC.FC     2 BARS
0911- 8D 88 FC
0914- FC                .HS 8D.88.FC.FC     2 BARS
0915- 8D 88 FC
0918- FC                .HS 8D.88.FC.FC     2 BARS
0919- 8D 88 FC
091C- FC                .HS 8D.88.FC.FC     2 BARS
091D- 8D 88 FC
0920- FC                .HS 8D.88.FC.FC     2 BARS
0921- 8D 88 FC          .HS 8D.88.FC        1 BAR AT END OF LINE
0924-                   >FCL -              FULL CURRENT LINE WITH DASHES
0926-                   >POS 3,0            VTAB 3, HTAB 0
0928- 8F                .HS 8F              INVERSE MODE
0929-                   >CEN "Demonstration of Message Printer"
094A- 8E 8D 8D          .HS 8E.8D.8D        Normal mode
094D-                   >CEN "S-C Software Corporation"
0966- 8D                .HS 8D
0967-                   >CEN "P. O. Box 280300"
0978- 8D                .HS 8D
0979-                   >CEN "Dallas, TX 75228"
098A- 8D 8D 8D          .HS 8D.8D.8D
098D-                   >WIN 0,80,0,24      Full screen again
0992- 00                .HS 00              End of message
                *---------------------------------
0993-           MSG1    >POS 11,1           VTAB 11, HTAB 1
0995- 9D                .HS 9D              CLR EOL
0996-                   >STR "SELECT ONE:  "
09A3- 00                .HS 00              End of message
                *---------------------------------
09A4-           MSG2    >POS 24,1           VTAB 24, HTAB 1
09A6- 8F                .HS 8F        INVERSE MODE
09A7-                   >STR " <SPACE> FOR MENU, <RETURN> FOR MORE "
09CC- 8E 00             .HS 8E.00     NORMAL MODE, EOM
                *---------------------------------
09CE- 87 87 8D  MSG3    .HS 87.87.8D
09D1-                   >STR "Wasn't that a nice demonstration?"
09F2- 8D 00             .HS 8D.00
                *-------------------------------
09F4- 20 18 08  TT      JSR ANOTHER.MESSAGE.PRINTER
09F7- 8D 8D             .HS 8D.8D
09F9-                   >CEN "This is another way to print messages."
0A20-                   >HTAB 0
0A22-                   >STR "<<<"
0A25-                   >HTAB 99      TAB TO END OF LINE
0A27- 88 88             .HS 88.88     BACK UP TWO SPACES
0A29-                   >STR ">>>"
0A2C- 8D 00             .HS 8D.00
0A2E- 20 18 08          JSR ANOTHER.MESSAGE.PRINTER
0A31-                   >REP 10,M-$40       10 CARRIAGE RETURNS
0A34-                   >WIN 0,1,14,24
0A39-                   >REP 10,|
0A3C-                   >WIN 79,1,14,24
0A41-                   >REP 10,|
0A44-                   >WIN 0,80,13,24
0A49-                   >FCL _
0A4B-                   >FIL 23,-
0A4E-                   >WIN 0,80,0,24
0A53-                   >VTAB 24
0A55- 00                .HS 00              End of message
0A56- 60                RTS
                *-------------------------------
```

Revision to DOS 3.3 for UniDisk 3.5.................Bill Morgan

Since we published my patches to allow DOS 3.3 to use the UniDisk 3.5 last May and June I've been using it every day, and just loving all that space. After basking in readers' praise for such a useful piece of code, you can imagine how I felt when I first went to boot that disk on the IIgs and watched it crash into the dirt!

After a little detective work the cause is found: When we boot from the //e Protocol Converter Interface it passes slot*16 in the A- and X-registers and the slot number in location $58. In my boot code in FORMAT.UNIDISK I was using the A-register and the contents of $58 to set up my RWTS. Come to find out, the IIgs only gives us the slot*16 in the X-register, so we have to do a little more manipulation to create the values we need. Here is the revised code:

```
3080 BOOT   TXA
3082        PHA          save slot*16
3090        LSR          .divide
3092        LSR          ..by 16
3094        LSR          ...to get
3096        LSR          ....slot
```

To update the boot image on an existing disk you can use the instructions at the top of page 23 of the July '86 issue. Be sure to note the caution in the last paragraph of page 21 in that issue, about not installing a DOS image that already contains RAMdisk patches: I tripped over that again myself!

A PER-fect Tool for 65816 Position-Independent Code

..........Sandy Mossberg

Digging into the 65816 Apple IIgs has provided more entertainment than I've had since the introduction of the Apple II. So far, my programming efforts have been in the ProDOS 8 environment. When Apple gets its act together on ProDOS 16, writing position-independent code will become critical. Why not do a bit of preparing now? (The following techniques will apply to older Apples which have been upgraded with a 65802 cpu chip, also.)

Absolute internal references are the bugaboo of the position-independent programmer. (By the way, I am not referring to those individuals who program as well standing on their heads as they do flat on their backs or seated in a chair; I am referring to programs which run correctly no matter where they are loaded into memory, without any modifications to reflect the position in memory.) The three most common absolute internal references are:

(1) the JMP to a program location;
(2) the JSR to a subroutine location; and
(3) the reading from or writing to program text or data tables or variables.

The first problem can be solved simply by using the BRL instruction instead of JMP. For avoiding the second and third traps, the PER opcode is an ideal tool. The solutions outlined below pertain only to code and data within one 64K bank of memory. Solutions which handle programs and data in multiple banks will have to wait for future articles.

PER is one of the new opcodes found in the 65802 and 65816. PER stands for "Push Effective program counter (PC) Relative data word onto stack", and employs a hybrid stack/PC-relative addressing mode. PER adds the 16-bit signed displacement in its two-byte operand to the current value for the PC and pushes the result onto the stack. As usual, the high byte is pushed first, then the low byte. Regardless of the values for the m-, x-, and e-status bits, two bytes are always pushed onto the the stack. Because the operand is a displacement relative to the PC, the top two bytes of the stack always will point to the designated program code or data, irrespective of whether it is internal or external to the program. Thus, an internal JSR can be simulated by the following sequence:

```
                PER CONTINUE-1
                BRA SUBROUTINE
       CONTINUE ...
                ...
                ...
     SUBROUTINE ...
                ...
                RTS
```

Note that the continuation address to be placed on the stack must be one less than the desired address (CONTINUE-1 above),

because the RTS opcode increments the address it finds on the stack before jumping to that address. Also note that the BRA opcode works well if the subroutine is nearby (within about + or - 128 bytes of the BRA itself); if the subroutine is further away, you will need to use the BRL opcode here.

I have constructed two macros in the listing of the program which follows. >BSR calls nearby subroutines, and >BSL calls both nearby and distant subroutines. The macro definitions are in lines 1130-1210, and I have shown them in use in lines 1320-1330. The listed program itself is rather trivial, designed just to illustrate my contrived solutions. Lines 1270-1340 clear the screen and then print out the hex values from $00 to $F0, with an interval of $10. The two loops in lines 1350-1560 show two different ways to access data which is part of a program and still be position-independent; their function is to print out a string.

To access internal data, PER again comes to the rescue by allowing us to establish an external (zero page or stack) pointer to the data. Here, the operand must represent the exact address of the target data. The first example stores the pointer generated by PER into a pointer-variable in page zero (lines 1360-1400). PER puts the pointer onto the stack, and two PLAs pull it back off. If we were in Native mode with the m-bit cleared for 16-bit operation, only one PLA and STA would be needed. We could replace lines 1370-1400 with:

```
        CLC
        XCE                    ENTER NATIVE MODE
        REP #$20               M=0, 16-BIT DATA
        PLA                    PULL OFF BOTH BYTES
        STA PTR                STORE BOTH BYTES
        XCE                    BACK TO EMULATION MODE
```

The second approach is to leave the REP-generated pointer on the stack. Line 1500 shows how we can use it with stack-relative indirect-indexed address mode to access the same text string. Lines 1550 and 1560 are then absolutely necessary, to remove the pointer from the stack when we are finished with it.

Though it is not appropriate for this sample program, the power of native mode is evident when working with 16-bit address pointers. The problem of using native mode in simple examples is that you cannot call the old standard monitor entry points (such as HOME, COUT, and PRBYTE) from native mode.

If you are going to write code which will be crossing 64K banks, you can still use the PER instruction. However, you also have to take into account the program bank and data bank registers. More on this at another time.

```
                   1000 *SAVE S.POSITION
                   1010 *--------------------------------
                   1020 *   POSITION INDEPENDENT CODE WITH THE 65802/16
                   1030 *--------------------------------
                   1040        .OP 65816
                   1050 *--------------------------------
 06-               1060 NUM    .EQ $06
 07-               1070 PTR    .EQ $07,08
                   1080 *--------------------------------
```

```
FC58-                 1090 HOME   .EQ $FC58
FDDA-                 1100 PRBYTE .EQ $FDDA
FDED-                 1110 COUT   .EQ $FDED
                      1120 *--------------------------------
                      1130        .MA BSR      BRANCH RELATIVE TO SUBROUTINE
                      1140        PER :1+1     When Subroutine is Nearby
                      1150 :1     BRA ]1
                      1160        .EM
                      1170 *--------------------------------
                      1180        .MA BSL      BRANCH RELATIVE TO SUBROUTINE
                      1190        PER :1+2     When Subroutine is Far Away
                      1200 :1     BRL ]1
                      1210        .EM
                      1220 *--------------------------------
                      1230 TRIVIAL.EXAMPLE
000800- 38            1240        SEC          SET EMULATION MODE
000801- FB            1250        XCE          IN CASE CALLED FROM NATIVE MODE
000802- 08            1260        PHP          SAVE E-BIT
                      1270 *---Show Position-Independent subroutine calls--------
000803- 20 58 FC      1280        JSR HOME     CLEAR SCREEN
000806- A9 00         1290        LDA #0       FOR NUM = 0 TO $F0 STEP $10
000808- 85 06         1300 .1     STA NUM
00080A- 20 DA FD      1310        JSR PRBYTE   Print in hexadecimal
00080D-               1320        >BSL PRBLNK  Print two blanks
00080D- 62 02 00      0000>        PER :1+2     When Subroutine is Far Away
000810- 82 36 01      0000> :1     BRL PRBLNK
000813-               1330        >BSR ADD.10  ...NEXT NUM
000813- 62 01 00      0000>        PER :1+1     When Subroutine is Nearby
000816- 80 2B         0000> :1     BRA ADD.10
000818- 90 EE         1340        BCC .1       ...NOT FINISHED YET
                      1350 *---Show Page-Zero Indirect data access--------------
00081A- 62 34 01      1360        PER TEXT     PUT POINTER TO TEXT ON STACK
00081D- 68            1370        PLA          Now put pointer into page zero
00081E- 85 07         1380        STA PTR
000820- 68            1390        PLA
000821- 85 08         1400        STA PTR+1
000823- A0 00         1410        LDY #0       Point to beginning of text string
000825- B1 07         1420 .2     LDA (PTR),Y  Get next character of string
000827- F0 06         1430        BEQ .3       End of string
000829- 20 ED FD      1440        JSR COUT
00082C- C8            1450        INY
00082D- D0 F6         1460        BNE .2       ...ALWAYS
                      1470 *---Show Stack-Indirect data access-----------------
00082F- 62 1F 01      1480 .3     PER TEXT     POINTER TO TEXT ON STACK
000832- A0 00         1490        LDY #0       Point to beginning of text string
000834- B3 01         1500 .4     LDA (1,S),Y  Get next character of string
000836- F0 06         1510        BEQ .5       End of string
000838- 20 ED FD      1520        JSR COUT
00083B- C8            1530        INY
00083C- D0 F6         1540        BNE .4       ...ALWAYS
00083E- 68            1550 .5     PLA          POP OFF TEXT POINTER
00083F- 68            1560        PLA
                      1570 *---Return to caller-------------
000840- 28            1580        PLP          GET SAVED E-BIT
000841- FB            1590        XCE          RESTORE E
000842- 60            1600        RTS
                      1610 *--------------------------------
000843- 18            1620 ADD.10 CLC
000844- A5 06         1630        LDA NUM      ADD $10 TO NUM
000846- 69 10         1640        ADC #$10     SET CARRY IF WRAPS AROUND
000848- 60            1650        RTS
                      1660 *--------------------------------
000849-               1670        .BS 256      FILLER TO MOVE PRBLNK FURTHER AWAY
                      1680 *--------------------------------
000949- A9 A0         1690 PRBLNK LDA #" "
00094B- 20 ED FD      1700        JSR COUT
00094E- 4C ED FD      1710        JMP COUT
                      1720 *--------------------------------
000951- 8D            1730 TEXT   .HS 8D
000952- D0 EF F3 E9
000956- F4 E9 EF EE
00095A- AD C9 EE E4
00095E- E5 F0 E5 EE
000962- E4 E5 EE F4
000966- A0 D4 E5 F8
00096A- F4 A0 D3 F4
00096E- F2 E9 EE E7 1740        .AS -/Position-Independent Text String/
000972- 00            1750        .HS 00
                      1760 *--------------------------------
```

Simple Line-Input Subroutine.............Bob Sander-Cederlof

The other day I needed a simple subroutine to read a line of text into an assembly language program. The routine had to be short, work in both 40- and 80-columns, and allow very simple editing. It had to read directly from the keyboard, regardless of where the input vector at KSWL ($38,39) might be pointing.

The following listing shows the result. READ.A.LINE is indeed short and fulfills all the needs I had. Backspace is the only editing character. When you type a printing character, it is displayed and the underline cursor advances. When you type a backspace, the cursor backs up and replaces what was previously at that position.

When you type a RETURN, the routine returns with the X-register pointing into the line buffer at the position where the RETURN is stored. If you type an ESCAPE, it nulls the line you just typed. The line is still in the buffer, but READ.A.LINE returns with the X-register equal to zero.

Inside READ.A.LINE I use the X-register to track position in the line buffer. I don't explicitly keep track of position on the screen, at least in absolute terms. Wherever the cursor is when you call READ.A.LINE is the place where the first cursor will be displayed. Since I do all moving around on the screen by simply printing out characters or backspacing, all screen positions are relative to the beginning position. You can use READ.A.LINE inside a small window if you like, and the input line will wrap nicely when it comes to the edge of the window.

Lines 1100-1120 handle the cursor by first printing an underline, and backspacing over it. Then when you type a character I will put it over the top of the cursor, and come back to line 1100 to display the next cursor.

Lines 1130-1150 are the standard way to read the keyboard when you don't want any firmware interference. (The Apple IIgs has more hardware bells and whistles here, with the Front Desk Processor and all, but this method will still work.)

When you type a backspace, I have to output a space to erase the cursor, then backspace twice and place a new cursor over the top of the character which was right before the old cursor. This is in lines 1350-1380. If the cursor is already at the beginning of the line, lines 1330-1340 will simply start over. Because of the way lines 1330 and 1340 are written, the maximum length line you can type and still use backspace is 128 characters.

I wrote a quick and dirty test program, to illustrate how to use READ.A.LINE. Lines 1500-1560 print out a prompting message. Line 1580 calls READ.A.LINE. The cursor will start right after the prompting message. Line 1600 tests for an ESCAPE, signifying a null entry. If you type RETURN, lines 1610-1670 will display the message you just entered.

```
                 1000 *SAVE S.READLINE
                 1010 *--------------------------------
C000-            1020 KEYBOARD   .EQ $C000
C010-            1030 STROBE     .EQ $C010
                 1040 *--------------------------------
FD8E-            1050 CROUT      .EQ $FD8E
FDED-            1060 COUT       .EQ $FDED
                 1070 *--------------------------------
                 1080 READ.A.LINE
0800- A2 00      1090 .0       LDX #0       Start at beginning of line
0802- A9 DF      1100 .1       LDA #"_"     Use underline for a cursor
0804- 20 ED FD   1110          JSR COUT     Print the Underline
0807- 20 49 08   1120          JSR BSOUT    ...and back up over it
080A- AD 00 C0   1130 .2       LDA KEYBOARD
080D- 10 FB      1140          BPL .2       ...wait until key pressed
080F- 8D 10 C0   1150          STA STROBE
0812- 9D 4E 08   1160          STA WBUF,X
0815- C9 A0      1170          CMP #$A0
0817- 90 0A      1180          BCC .4       CONTROL CHAR
0819- E0 7F      1190          CPX #MAXLEN
081B- B0 ED      1200          BCS .2
081D- 20 ED FD   1210          JSR COUT  SHOW IT ON SCREEN
0820- E8         1220          INX
0821- D0 DF      1230          BNE .1       ...ALWAYS
0823- C9 88      1240 .4       CMP #$88
0825- F0 0D      1250          BEQ .5       BACKSPACE
0827- C9 9B      1260          CMP #$9B
0829- F0 18      1270          BEQ .6       ESCAPE -- ABORT INPUT
082B- C9 8D      1280          CMP #$8D
082D- D0 DB      1290          BNE .2       ...NOT <RETURN>, IGNORE IT
082F- 20 46 08   1300          JSR SPCOUT   ERASE CURSOR FROM SCREEN
0832- E8         1310          INX          INCLUDE <RETURN> IN X-VALUE
0833- 60         1320          RTS          RETURN TO CALLER
0834- CA         1330 .5       DEX
0835- 30 C9      1340          BMI .0       ...ALREADY AT BEGINNING OF LINE
0837- 20 46 08   1350          JSR SPCOUT     ...SPACE
083A- 20 49 08   1360          JSR BSOUT      ...BACKSPACE
083D- 20 49 08   1370          JSR BSOUT      ...AND ANOTHER
0840- 4C 02 08   1380          JMP .1
0843- A2 00      1390 .6       LDX #0       SIGNAL ESCAPE TYPED
0845- 60         1400          RTS
                 1410 *--------------------------------
0846- A9 A0      1420 SPCOUT LDA #" "       SPACE
0848- 2C         1430          .HS 2C
0849- A9 88      1440 BSOUT  LDA #$88       BACKSPACE
084B- 4C ED FD   1450          JMP COUT
                 1460 *--------------------------------
7F-              1470 MAXLEN     .EQ 127
084E-            1480 WBUF       .BS MAXLEN+1
                 1490 *--------------------------------
                 1500 TEST
08CE- A2 00      1510          LDX #0       PRINT OUT A PROMPT MESSAGE
08D0- BD F1 08   1520 .0       LDA PROMPT,X
08D3- 20 ED FD   1530          JSR COUT
08D6- E8         1540          INX
08D7- E0 0F      1550          CPX #PROMPT.LEN
08D9- 90 F5      1560          BCC .0
                 1570 *--------------------------------
08DB- 20 00 08   1580          JSR READ.A.LINE
                 1590 *--------------------------------
08DE- F0 10      1600          BEQ .2       ESCAPE ABORTED IT
08E0- 20 8E FD   1610          JSR CROUT    <RETURN> OUTPUT
08E3- A2 00      1620          LDX #0       PRINT OUT THE LINE YOU TYPED IN
08E5- BD 4E 08   1630 .1       LDA WBUF,X
08E8- 20 ED FD   1640          JSR COUT
08EB- E8         1650          INX
08EC- C9 8D      1660          CMP #$8D     UP TO A <RETURN>
08EE- D0 F5      1670          BNE .1
08F0- 60         1680 .2       RTS
                 1690 *--------------------------------
08F1- C5 EE F4
08F4- E5 F2 A0
08F7- E1 A0 EC
08FA- E9 EE E5
08FD- BA A0 A0   1700 PROMPT .AS -/Enter a line:  /
0F-              1710 PROMPT.LEN .EQ *-PROMPT
                 1720 *--------------------------------
```

Friendly Decimal Print Subroutine..............Jan Eugenides

Several years ago Bob S-C published a little subroutine to convert two-byte hexadecimal values to decimal and print them. I have written a similar subroutine, but mine allows you to print any character as a leading character when the number has less than five digits, or to have no leading characters at all. Bob's subroutine always printed leading zeroes.

To use my subroutine, you put the number in the A- and X-registers (lo-byte in A, hi-byte in X), and the fill character in the Y-register. If you do not want any fill character, call my subroutine with Y=0. To make calling it easier, Bob S-C constructed a macro named PDEC, defined in lines 1490-1580 below. To print a number with no fill character, simply use the macro like this: >PDEC VALUE; to print with a fill character, use the macro with the fill character as the second parameter like this: >PDEC VALUE,"#".

Now let's look at my subroutine. Lines 1110-1120 store the high byte of the number, and the fill character. I waited to store the low-byte of the number in line 1160, inside the conversion loop, to save a few bytes and cycles.

Lines 1150-1380 comprise a loop controlled by X, counting down from 3 to 0. The loop prints out the first four digits and/or fill characters. X indexes into a power-of-tens table. I first subtract 10000 as many times as possible, counting each subtraction in the Y-register. Since Y starts out as ASCII 0, it increments through successive ASCII digits. When the remainder is finally less than 10000, I print the first digit. Then the process is repeated for 1000, 100, and 10.

Leading-zeroes may be either suppressed, or printed with your choice of fill character. The work is done in lines 1280-1340. I keep a flag in bit 7 of FLAG. This bit starts out 0 (see line 1140). As soon as I get a non-zero digit, I change the flag bit to 1 (see lines 1280-1300). If the flag bit is 1, lines 1310-1320 will decide to print the digit regardless of its value. (We don't want to omit or replace ALL zeroes, only LEADING zeroes.) Lines 1330-1340 will decide whther to print a fill character or nothing at all.

After printing the ten's digit, the remainder in NUM is the units digit in binary. All I have to do is add an ASCII "0" to change this remainder to ASCII, which I do in lines 1390-1400. By handling the unit's digit in this special way I save extra logic which would have been required to make the value $0000 print at least one zero and two bytes in my power-of-tens table, saving a total of six bytes. The cost is lines 1390 and 1400 versus a simple RTS, or four bytes. This method is also faster, because I don't have to cycle though the loop counting down the unit's digit.

For an exercise, you might try extending my subroutine to convert 3- or 4-byte values. It is really not too difficult. Of course, you would no longer be able to pass the value to be converted in the A- and X-registers. For 3-byte values, the

power-of-ten tables would have to be extended in both dimensions:

```
TENTBL .DA #10,#100,#1000,#10000
       .DA #100000,#1000000,#10000000
TENTBM .DA /10,/100,/1000,/10000
       .DA /100000,/1000000,/10000000
TENTBH .DA #0,#0,#0,#0
       .DA /100000/256,/1000000/256,/10000000/256
```

You would also have to extend the comparison and subtraction code inside the loop. Why not try it?

```
                *SAVE S.PRINT.DECIMAL
                *--------------------------------
                *   By Jan Eugenides and Bob S-C
                *
                *      Call with A, X, and Y as follows:
                *         (A) = low-byte of number to be printed
                *         (X) = high byte of number
                *         (Y) = fill character (or 00 if no fill)
                *--------------------------------
FDED-           COUT   .EQ $FDED
                *--------------------------------
0800- 8E 52 08  PRDEC  STX NUM+1     Store high byte of number
0803- 8C 50 08         STY FILL      Store fill character
0806- A2 03            LDX #3        FOR X = 3 TO 0
0808- 8E 4F 08         STX FLAG      Clear bit 7 in leading-zero flag
080B- A0 B0     .1     LDY #"0"      Start with digit = ASCII 0
080D- 8D 51 08  .2     STA NUM       Compare number to power of ten
0810- DD 47 08         CMP TENTBL,X      10^(X+1) ... lo-byte
0813- AD 52 08         LDA NUM+1
0816- FD 4B 08         SBC TENTBH,X      10^(X+1) ... hi-byte
0819- 90 0C            BCC .3        Remainder is smaller than 10^(X+1)
081B- 8D 52 08         STA NUM+1     Store remainder hi-byte
081E- AD 51 08         LDA NUM       Get remainder lo-byte
0821- FD 47 08         SBC TENTBL,X
0824- C8               INY           Increment ASCII digit
0825- D0 E6            BNE .2        ...always
                *---Print a digit----------------
0827- 98        .3     TYA           digit in ASCII
0828- C9 B0            CMP #"0"      Is it a zero?
082A- F0 03            BEQ .4        ...yes, might be leading zero
082C- 8D 4F 08         STA FLAG      ...no, so clear leading-zero flag
082F- 2C 4F 08  .4     BIT FLAG      If this is leading-zero, will be +
0832- 30 05            BMI .5        ...not a leading zero
0834- AD 50 08         LDA FILL      ...leading zero, so use fill-char
0837- F0 03            BEQ .6        ...Oops, no fill-char
0839- 20 ED FD  .5     JSR COUT      Print the digit or fill-char
083C- AD 51 08  .6     LDA NUM       Get lo-byte of remainder
083F- CA               DEX           Next X
0840- 10 C9            BPL .1        Go get next digit
0842- 09 B0            ORA #"0"      Change remainder to ASCII
0844- 4C ED FD         JMP COUT      Print Unit's digit & RTS
                *--------------------------------
0847- 0A 64 E8
084A- 10        TENTBL .DA #10,#100,#1000,#10000
084B- 00 00 03
084E- 27        TENTBH .DA /10,/100,/1000,/10000
                *--------------------------------
084F-           FLAG   .BS 1
0850-           FILL   .BS 1
0851-           NUM    .BS 2
                *--------------------------------
                       .ma pdec
                       lda ]1
                       ldx ]1+1
                       .do ]#>1
                       ldy #"]2"
                       .else
                       ldy #0
                       .fin
                       jsr prdec
                       .em
```

```
                    1590 *--------------------------------
FD8E-               1600 CROUT  .EQ $FD8E
                    1610
0853-               1620 T       >PDEC VAL1,#
0853- AD 9A 08      0000>         lda VAL1
0856- AE 9B 08      0000>         ldx VAL1+1
                    0000>         .do 2>1
0859- A0 A3         0000>         ldy #"#"
                    0000>         .else
                    0000>         .fin
085B- 20 00 08      0000>         jsr prdec
085E- 20 8E FD      1630         JSR CROUT
0861-               1640         >PDEC VAL2,^
0861- AD 9C 08      0000>         lda VAL2
0864- AE 9D 08      0000>         ldx VAL2+1
                    0000>         .do 2>1
0867- A0 DE         0000>         ldy #"^"
                    0000>         .else
                    0000>         .fin
0869- 20 00 08      0000>         jsr prdec
086C- 20 8E FD      1650         JSR CROUT
086F-               1660         >PDEC VAL3
086F- AD 9E 08      0000>         lda VAL3
0872- AE 9F 08      0000>         ldx VAL3+1
                    0000>         .do 1>1
                    0000>         .else
0875- A0 00         0000>         ldy #0
                    0000>         .fin
0877- 20 00 08      0000>         jsr prdec
087A- 20 8E FD      1670         JSR CROUT
087D-               1680         >PDEC VAL4
087D- AD A0 08      0000>         lda VAL4
0880- AE A1 08      0000>         ldx VAL4+1
                    0000>         .do 1>1
                    0000>         .else
0883- A0 00         0000>         ldy #0
                    0000>         .fin
0885- 20 00 08      0000>         jsr prdec
0888- 20 8E FD      1690         JSR CROUT
088B-               1700         >PDEC VAL5," "
088B- AD A2 08      0000>         lda VAL5
088E- AE A3 08      0000>         ldx VAL5+1
                    0000>         .do 2>1
0891- A0 A0         0000>         ldy #" "
                    0000>         .else
                    0000>         .fin
0893- 20 00 08      0000>         jsr prdec
0896- 20 8E FD      1710         JSR CROUT
0899- 60            1720         RTS
                    1730
089A- D2 04         1740 VAL1    .DA 1234
089C- 4F 00         1750 VAL2    .DA 79
089E- FF FF         1760 VAL3    .DA 65535
08A0- 4B 1B         1770 VAL4    .DA 6987
08A2- 17 00         1780 VAL5    .DA 23
```

Note from American Foundation for the Blind

The American Foundation for the Blind is managing a database of individuals with hands-on experience with computers, low vision aids, talking products, and other devices for the blind and visually-impaired. About 500 users and evaluators are currently listed in their database, and are available to help in training, purchase decisions, and evaluation of new products. If you have related experience are interested in helping, or could use such help yourself, you can contact their National Technology Center at 1-800-232-5463.

Some Handy MOVE Macros...................Bob Sander-Cederlof

Some time ago Bruce Love, a mathematics teacher at Hillcrest High School in New Zealand, sent me a macro he has found quite handy. It is shown in lines 1010-1090 in the listing below. It can be used to generate in-line loops to copy a group of bytes from one place to another. The macro works with any group up to 256 bytes long. It is really quite flexible, as the several examples in lines 1460-1550 reveal. Note that the source and destination may be either absolute or indirect in form. Furthermore, the length may either be stored in a variable or in the immediate form.

I added the SET macro (lines 1170-1230) to set up a pointer, to simplify using the MOVE macro with pointers in page zero. An example of its use is in lines 1490 and 1500.

There is another macro that I have found useful for moving larger chunks of RAM around. I use it especially in ProDOS SYSTEM files for relocating program sections. See lines 1100-1160 for the definition of the COPY macro, which moves up to 256 pages from source to destination. The parameters for COPY are reminiscent of the monitor M-command. For example, to copy $4B00 through $5FFF to a destination beginning at $6A00 the monitor command would be "6A00<4B00.5FFFM". The equivalent COPY macro would be "COPY $6A00,$4B00,$5FFF".

With both of these macros, beware of overlapping the range of the source and destination just as you would with the monitor M-command. Unless, of course, you know what you are doing and are doing it on purpose. For example, you can copy the first character of a block to all positions in the block like this:

```
        >MOVE BUFFER,BUFFER+1,LEN
```

As is usually the case with macros which generate code, the code will probably not be as efficient as you could write given more knowledge of the overall situation. However, macros like these can shorten the time to develop programs, and thus they are worthy in spite of their possible inefficiencies.

```
1000 *SAVE MOVE.MACROS
1010 *--------------------------------
1020        .MA MOVE source,destination,length
1030        LDY #0
1040 :1     LDA ]1,Y        GET SOURCE BYTE
1050        STA ]2,Y        STORE DEST. BYTE
1060        INY             NEXT BYTE
1070        CPY ]3
1080        BCC :1          ...IF MORE
1090        .EM
1100 *--------------------------------
1110        .MA COPY destination,src.start,src.end
1120        LDA /]1         DESTINATION PAGE
1130        LDY /]2         SOURCE BEGINNING PAGE
1140        LDX /]3-]2+256       # OF PAGES
1150        JSR COPY.PAGES
1160        .EM
1170 *--------------------------------
1180        .MA SET pointer,value
1190        LDA #]2
1200        STA ]1
1210        LDA /]2
1220        STA ]1+1
1230        .EM
```

```
              *--------------------------------
00-             SPTR   .EQ $00,01
02-             DPTR   .EQ $02,03
04-             LENGTH .EQ $04
0200-           BUFFER .EQ $200
              *--------------------------------
                COPY.PAGES
0800- 85 03            STA DPTR+1
0802- 84 01            STY SPTR+1
0804- A0 00            LDY #0
0806- 84 02            STY DPTR
0808- 84 00            STY SPTR
080A- B1 00     .1     LDA (SPTR),Y
080C- 91 02            STA (DPTR),Y
080E- C8               INY           NEXT BYTE
080F- D0 F9            BNE .1        ...KEEP GOING
0811- E6 01            INC SPTR+1    ...END OF PAGE
0813- E6 03            INC DPTR+1
0815- CA               DEX           NEXT PAGE
0816- D0 F2            BNE .1        ...KEEP GOING
0818- 60               RTS           ...FINISHED
              *--------------------------------
                SAMPLE.MOVES
0819-                  >MOVE $500,$600,#40
0819- A0 00     0000>   LDY #0
081B- B9 00 05  0000> :1 LDA $500,Y     GET SOURCE BYTE
081E- 99 00 06  0000>   STA $600,Y     STORE DEST. BYTE
0821- C8        0000>   INY           NEXT BYTE
0822- C0 28     0000>   CPY #40
0824- 90 F5     0000>   BCC :1        ...IF MORE
              *--------------------------------
0826-                  >SET SPTR,BUFFER+10
0826- A9 0A     0000>   LDA #BUFFER+10
0828- 85 00     0000>   STA SPTR
082A- A9 02     0000>   LDA /BUFFER+10
082C- 85 01     0000>   STA SPTR+1
082E-                  >SET DPTR,BUFFER+40
082E- A9 28     0000>   LDA #BUFFER+40
0830- 85 02     0000>   STA DPTR
0832- A9 02     0000>   LDA /BUFFER+40
0834- 85 03     0000>   STA DPTR+1
0836- A9 09            LDA #9
0838- 85 04            STA LENGTH
083A-                  >MOVE (SPTR),(DPTR),LENGTH
083A- A0 00     0000>   LDY #0
083C- B1 00     0000> :1 LDA (SPTR),Y     GET SOURCE BYTE
083E- 91 02     0000>   STA (DPTR),Y     STORE DEST. BYTE
0840- C8        0000>   INY           NEXT BYTE
0841- C4 04     0000>   CPY LENGTH
0843- 90 F7     0000>   BCC :1        ...IF MORE
              *--------------------------------
0845-                  >MOVE BUFFER,(DPTR),#10
0845- A0 00     0000>   LDY #0
0847- B9 00 02  0000> :1 LDA BUFFER,Y     GET SOURCE BYTE
084A- 91 02     0000>   STA (DPTR),Y     STORE DEST. BYTE
084C- C8        0000>   INY           NEXT BYTE
084D- C0 0A     0000>   CPY #10
084F- 90 F6     0000>   BCC :1        ...IF MORE
              *--------------------------------
                SAMPLE.COPIES
0851-                  >COPY $6A00,$4B00,$5FFF
0851- A9 6A     0000>   LDA /$6A00       DESTINATION PAGE
0853- A0 4B     0000>   LDY /$4B00       SOURCE BEGINNING PAGE
0855- A2 15     0000>   LDX /$5FFF-$4B00+256      # OF PAGES
0857- 20 00 08  0000>   JSR COPY.PAGES
085A-                  >COPY $5000,$2200,$4AFF
085A- A9 50     0000>   LDA /$5000       DESTINATION PAGE
085C- A0 22     0000>   LDY /$2200       SOURCE BEGINNING PAGE
085E- A2 29     0000>   LDX /$4AFF-$2200+256      # OF PAGES
0860- 20 00 08  0000>   JSR COPY.PAGES
0863-                  >COPY $1000,BUFFER,BUFFER+255
0863- A9 10     0000>   LDA /$1000       DESTINATION PAGE
0865- A0 02     0000>   LDY /BUFFER       SOURCE BEGINNING PAGE
0867- A2 01     0000>   LDX /BUFFER+255-BUFFER+256      # OF PAGES
0869- 20 00 08  0000>   JSR COPY.PAGES
086C- 60               RTS
              *--------------------------------
```

New Supplement to "Beneath Apple ProDOS" Available

Quality Software has published a new supplement to for "Beneath Apple ProDOS," which includes information on ProDOS versions 1.2 and 1.3. It is 30 pages longer than the previous edition, which covered version 1.1.1 of ProDOS.

You might be wondering, "What is a supplement, anyway?" The book "Beneath Apple ProDOS" ("BAP") contains much reference material needed to really take advantage of ProDOS capabilities. While other books now cover much of the same ground, "BAP" was the first one to put it all into print. The supplement, however, is unique: it contains a complete description of the internal details of both the ProDOS MLI kernel and BASIC.SYSTEM. If you are at all involved with the inner works of ProDOS, or are having trouble finding out the REAL scoop on some issues, you NEED the supplement. I have used my copies extensively, and depended heavily on it when writing the ProDOS version of S-C Macro Assembler.

The original supplement, for versions 1.0.1 and 1.0.2 cost $10. The second and third editions are $12.50 each. Incredibly low-priced! You must order these directly from Quality Software, at 21610 Lassen Street #7, Chatsworth, CA 91311. As I understand it, the supplement is only sold to owners of "BAP", and you have to use the coupon found on page 8-9 of "BAP" to do the ordering. ("BAP" itself is $19.95 retail, but we sell it for $18 here.)

A Note about Chessmaster 2000

Gina Montagano, of the Software Toolworks, recently sent me "Chessmaster 2000". If you have not heard of it, Chessmaster is a really good chess-playing program which will run on the Apple II family. It is also available for Mac, IBM, and others. I bought Microchess back in 1977, and Sargon in 1978; they did well, considering they ran in such small machines. Microchess would fit in a 4K Apple! We have come a long way since then. Chessmaster has many nice features, is the current reigning "champion", and is well worth the $49.95 price. (My 11-year-old stepson Matthew figured out a neat way to beat it. He let it tromp all over him, all the way to checkmate. Then he stepped back one move, swapped sides, and won!) Some features: display board in color or monochrome, 2-D or 3-D; 20 levels of competence, plus one that purposely makes occasional obviously bad moves; optional hints for your best move; huge opening book; library of famous games; and on and on. Negative: copy protected, although you may order a backup copy for $5. Chessmaster 2000 is available at most software stores.

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are $1.80 each (other countries inquire for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.